14206

Using Support Vector Machines to Classify Whether a Car is in Front of You or Not

Technical Paper – Vetronics (In-House)
By Michael S. Del Rose

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **27 JUL 2004**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**
4. TITLE AND SUBTITLE **Using Support Vector Machines to Classify Whether a Car is in Front of You or Not**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) **Michael S. Del Rose**		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **USA TACOM**		8. PERFORMING ORGANIZATION REPORT NUMBER **14206**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S) **TACOM TARDEC**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release, distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**SAR**	**16**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

1.0 Introduction

Support Vector Machine (SVM) theory is a learning machine theory developed by V. Vapnik. Its most common uses are for classification problems and regression. Like other learning machines, the distribution of the population does not need to be known. It is sufficient only to know that a distribution exists. What sets SVMs apart from other learning machines is its ability to classify items correctly with a relatively small sample size.

In this paper I will briefly describe learning machines and SVMs. It will not be a complete tutorial on either subject. If the reader desires to learn more about them, then please refer to the references at the end of this paper. I will also give the theory and results of using SVMs to classify whether there is a car in front of you or not, using an image from a digital camera.

2.0 Learning Machines

A learning machine is a process for creating an optimal algorithm that chooses one class over another based on a set of training data. Vapnik picturizes a learning machine similar to the one in figure 2.1.

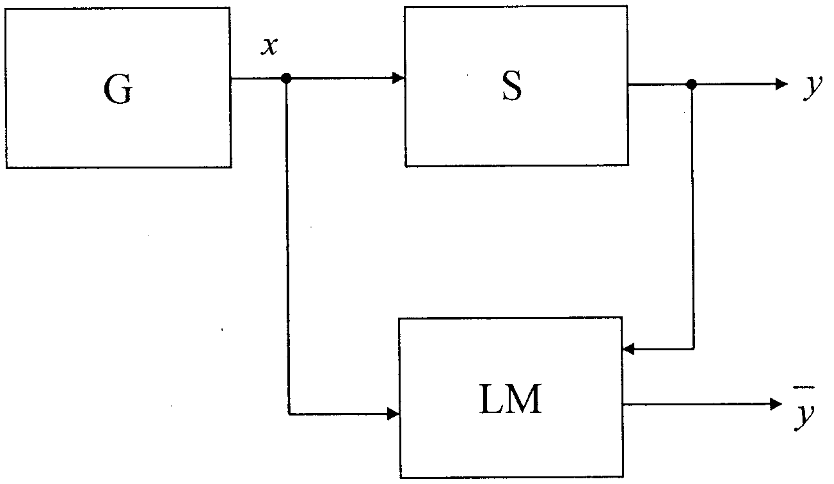

Figure 2.1

Here, G is the generator of the data. It is responsible for the distribution (but the distribution is unknown). S is the supervisor operator. It separates the data into classes. The LM is the learning machine. It takes in the data that was generated by x and what class it belongs to. From this information it creates an algorithm that is supposed to represent the real population classification. The output of the LM is calculation from the algorithm of what class the data belongs to. It is the hope that the learning machine's output and the supervisor's output are equal. The error is based on the differences of *y* and *y bar*. In other words, the LM is supposed to perform the following mapping:

$$\vec{x_i} \to y_i$$

Now, we can consider y_i a function of $\mathbf{x}_i$ and α where α is an adjustable parameter that describes a function in the set of all possible functions (see equation 2.1)

$$y_i = f(\vec{x}_i, \alpha) \qquad \textbf{(eq 2.1)}$$

The mathematical expectation of the test error is defined as:

$$R(\alpha) = \int \frac{1}{2} \left| y - f(\vec{x}, y) \right| dP(\vec{x}, y) \qquad \textbf{(eq 2.2)}$$

where P(*x*,*y*) is the joint probability distribution.

The goal of the learning machine is to minimize R(α). Since we can not determine the value of R(α) (we don't know the distribution of the population), we must bound R(α) and minimize the bound. We determine the empirical risk as:

$$R_{emp}(\alpha) = \frac{1}{l} \sum_{i=1}^{l} \frac{1}{2} \left| y_i - f(\vec{x}_i, \alpha) \right| \qquad \textbf{(eq 2.3)}$$

l is the number of observations. We can now define the bounds of the risk (see equation 2.4).

$$R(\alpha) \le R_{emp}(\alpha) + \phi\left(\frac{h}{l}, \frac{\log(\eta)}{l} \right) \qquad \textbf{(eq 2.4)}$$

Where the ϕ(...) is the VC confidence, h is the VC dimension, and η is chosen so that 0<η<1 and 1-η represents the confidence in the empirical risk. The VC dimension is defined as the maximum number of points that can be shattered by all possible equations. The VC confidence can be found in equation 2.5.

$$\phi\left(\frac{h}{l}, \frac{\log(\eta)}{l} \right) = \sqrt{\frac{h\left(\log\left(\frac{2l}{h} \right) + 1 \right) - \log\left(\frac{\eta}{4} \right)}{l}} \qquad \textbf{(eq 2.5)}$$

We now have a bound for the risk, R(α), which can be computed.

3.1 Support Vector Machines

The idea behind many classification problems is finding separating curves that divide data between their classes. support vector machines divide the data between two classes. The value for one class is represented as 1 (y = 1 in figure 2.1) and the value of the other class is represented as -1.

3.2 Linear Separable Case

Figure 3.1 shows a linear separable set of data between two classes. Let's call class 1 the set where the data points (the *s) are to the upper right of the separating line. Call the data points (the Xs) to the lower left of the separating line class -1.

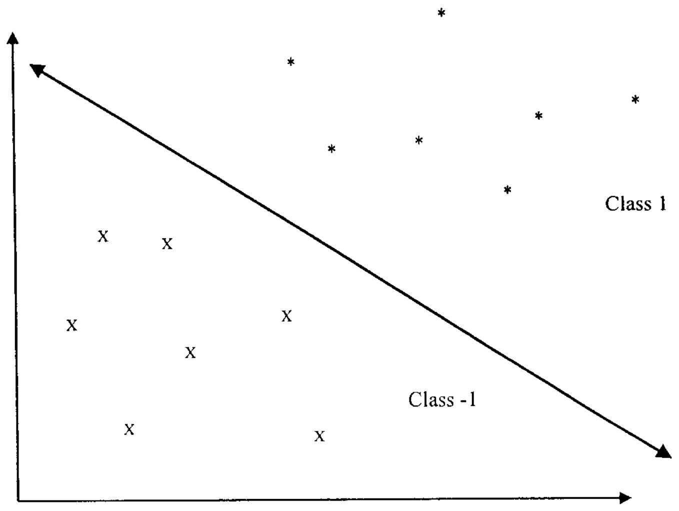

Figure 3.1

Our goal is to maximize the distance between the separating line and the each data set. The following steps accomplish this:

1.) Draw a convex hull around each data set.

2.) Calculate the distance between each point in one class and each side of the convex hull in the opposing class.

3.) Choose the line, which makes up the smallest distance in part 2. This line will be called the 'Minimum Distance Line' or MDL.

4.) Calculate a line that intersects the midpoint of the MDL and is parallel to the side of the hull used for calculating the MDL. This line is the separating line.

figure 3.2 shows this process.

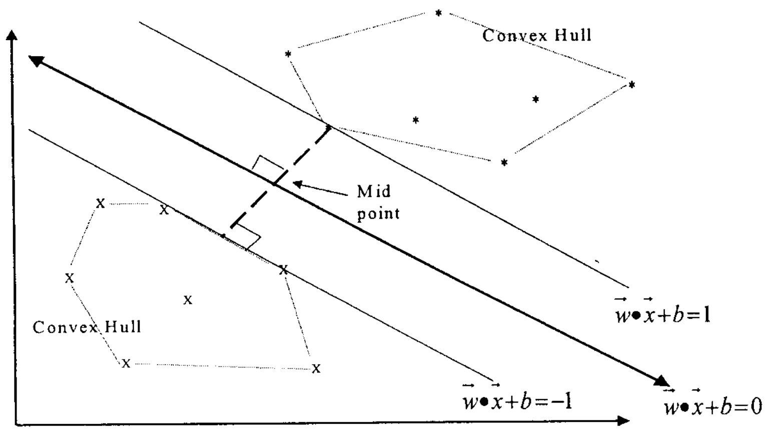
Convex Hull
Mid point
Convex Hull
$\vec{w} \bullet \vec{x}+b=1$
$\vec{w} \bullet \vec{x}+b=-1$
$\vec{w} \bullet \vec{x}+b=0$

Figure 3.2

Notice that there are three points (in this case) that determine the separating line: 1 from class 1 and 2 from class -1. This three points are considered the support vectors.

The equation for the separating line is

$$\vec{w} \bullet \vec{x} + b = 0 \qquad \textbf{(eq 3.1)}$$

w is the perpendicular vector from the separating line and b is the offset.

The dividing line for class 1 is eq 3.2 where x_1 is the data set for class 1. The dividing line for class -1 is eq 3.3 where x_2 is the data set for class -1.

$$\vec{w} \bullet \vec{x_1} + b = 1 \qquad \textbf{(eq 3.2)}$$

$$\vec{w} \bullet \vec{x_2} + b = -1 \qquad \textbf{(eq 3.3)}$$

The data values for class 1 follow the inequality of eq 3.4 and class -1 follows eq 3.4.

$$\vec{w} \bullet \vec{x} + b \geq 1 \qquad \textbf{(eq 3.4)}$$

$$\vec{w} \bullet \vec{x} + b \leq -1 \qquad \textbf{(eq 3.5)}$$

Since y_i is 1 for class 1 and -1 for class -1, eq 3.4 and eq 3.5 can be combined into eq. 3.6

$$y_i\left(\left(\vec{w} \bullet \vec{x_i}\right) + b\right) \geq 1 \qquad \textbf{(eq 3.6)}$$

From eq 3.2 and eq 3.3, we can combine them and normalize to get eq 3.7

$$\frac{\vec{w}}{\left\|\vec{w}\right\|}\left(\vec{x_1} - \vec{x_2}\right) = \frac{2}{\left\|\vec{w}\right\|} \qquad \textbf{(eq 3.7)}$$

The optimal dividing line will be the line that maximizes the right side of eq 3.7. Or, in other words, we want to minimize equation 3.8 with eq 3.6 as one of the constraints.

$$\frac{1}{2}\left\|\vec{w}\right\|^2 \qquad \textbf{(eq 3.8)}$$

Optimization is done using LaGrange multipliers (α_i), minimizing the new equation with respect to the primal variables w and b. Call this equation L(w,b,α), or L for short. The main idea is to find the saddle point of L. The KKT (Karush, Khan, Tucker) condition states that the saddle point appears where the primal variables vanish, i.e.

$$\frac{\partial}{\partial b} L(\vec{w}, b, \vec{\alpha}) = 0$$

$$\frac{\partial}{\partial b} L(\vec{w}, b, \vec{\alpha}) = \frac{\partial}{\partial b}\left(\frac{1}{2}\left\|\vec{w}\right\|^2 - \sum_{i=1}^{l} \alpha_i \left(y_i\left(\vec{w} \bullet \vec{x_i}\right) + b\right) - 1\right) = 0$$

$$= \sum_{i=1}^{l} \alpha_i y_i = 0 \qquad \textbf{(eq 3.9)}$$

and

$$\frac{\partial}{\partial \vec{w}} L(\vec{w}, b, \vec{\alpha}) = 0$$

$$\frac{\partial}{\partial \vec{w}} L(\vec{w}, b, \vec{\alpha}) = \frac{\partial}{\partial \vec{w}}\left(\frac{1}{2}\left\|\vec{w}\right\|^2 - \sum_{i=1}^{l} \alpha_i \left(y_i\left(\vec{w} \bullet \vec{x_i}\right) + b\right) - 1\right) = 0$$

$$= \vec{w} - \sum_{i=1}^{l} \alpha_i y_i \vec{x_i} = 0$$

$$\Rightarrow \boxed{\vec{w} = \sum_{i=1}^{l} \alpha_i y_i \vec{x_i}} \qquad \textbf{(eq 3.10)}$$

So the solution vectors can be found where all values of α_i do not equal 0. These are the support vectors. Continuing on, we can substitute eq 3.9 and 3.10 into the equation for L and eliminate the primal variables (***w***, b). We are left with the Wolfe dual of the optimization
:

$$W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \left(\vec{x_i} \bullet \vec{x_j}\right) \qquad \textbf{(eq 3.11)}$$

subject to the following constraints:

$$\alpha_i \geq 0 \qquad \text{for i} = 1,2,\ldots,l \qquad \textbf{(eq 3.12)}$$

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \qquad \textbf{(eq 3.13)}$$

Finally, the decision function can be written as:

$$f(\vec{x}) = \operatorname{sgn}\left(\sum_{i=1}^{l} y_i \alpha_i \bullet (\vec{x} \bullet \vec{x}_i) + b\right) \qquad \textbf{(eq 3.14)}$$

or,

$$\boxed{f(\vec{x}) = \operatorname{sgn}(\vec{w} \bullet \vec{x} + b)} \qquad \textbf{(eq 3.15)}$$

where *w* is found by eq 3.10 and *b* is computed from:

$$\boxed{\alpha_i \left\lfloor y_i \left((\vec{x}_i \bullet \vec{w}_i) + b \right) - 1 \right\rfloor = 0} \qquad \textbf{(eq 3.16)}$$

3.3 Non-Linear Separable Case

In most real world classification problems, the data does not follow a simple linear separation. We are often forced into transforming the data set into a different space (called feature space) to accurately portray all the features before any learning can take place. This is very computer intensive for most learning machines, however, support vector machines do not need to transfer all the data into the feature space.

Our goal is to find a mapping function (called Φ) from the real space (called R^n) to the feature space (called F), i.e.:

$$\Phi : \Re^n \longrightarrow F$$

and cut down the time it takes to process. If we look at the equation for the optimal construction of the hyperplane in the feature space (eq 3.11) and the decision equation (eq. 3.14 or eq. 3.15) we can see that only dot products, i.e. $\Phi(x) \bullet \Phi(y)$, need to be manipulated. We don't have to transform all of our data, only the dot products of the data. We introduce the kernel in our equations: $k(x,y) = \Phi(x) \bullet \Phi(y)$.

Our decision function (eq. 3.14) now becomes

$$f(\vec{x}) = \operatorname{sgn}\left(\sum y_i \alpha_i \bullet k(\vec{x}, \vec{x}_i) + b\right) \qquad \textbf{(eq 3.17)}$$

and the Wolfe dual equation (from eq. 3.11) becomes

$$W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j k(\vec{x}_i, \vec{x}_j) \qquad \textbf{(eq 3.18)}$$

subject to the same constraints as before.

That's it! No transforming of data to feature space, only the evaluation of the dot products in feature space.

3.4 What About a Non-Separable Case?

The support vector machines can still perform the non-separable case with relatively good accuracy by relaxing the constraints. To do this we use a slack variable (called $\xi_i >= 0$ for i = 1,2,...,*l*). Equation 3.6 will become

$$y_i\left(\left(\vec{w} \bullet \vec{x_i}\right) + b\right) \geq 1 - \xi_i \qquad \textbf{(eq 3.19)}$$

The end result is the same as the non separable case except for the change of the constraint α_i:

$$0 \leq \alpha_i \leq C$$

where C is a value chosen to represent a limit on the error. In the separable case, we assume that C = infinity. Different values of C will change the separating line. This can be graphically shown in figure 3.3.

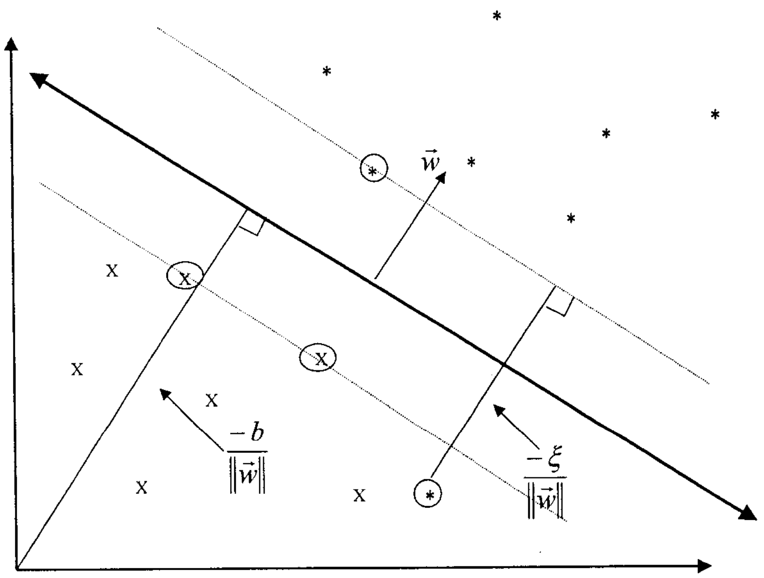

Figure 3.3

The circled points in figure 3.3 are the support vectors. The rest is self-explanatory.

4.1 The Classification of a Car in Front of You or Not

The data set used is a subset of a set from TACOM. In this set, pictures of cars were taken from (from inside the vehicle) in different areas and on different roads. The pictures were 512x640 with the car in the center (or not).

I took this data set and removed all cars that were in parking lots and dirt roads; leaving only pictures taken on a paved road. This was done so that I would not have to do a feature extraction of different textures of roads (dirt, parking lot, paved lane, etc...). The problem with identifying textures would be a very complex problem alone.

The data set was categorized into having a car in front or not. Cars based on a certain distance in front of the camera were used (estimated visually). Cars further out were categorized as a "no car in lane". The data set was then gray scaled and cut; I took the center 256x256 image (deleting 128 on each side and 192 on the top and bottom). Finally, the standard deviation was taken of each row and put into a vector. This is our feature vector that will be used to teach (and test) the support vector machine - it is a vector of 256 elements.

The data set I used had 75 pictures classified with a car in front and 54 pictures with no car in front (see figure 4.1 and 4.2 for examples of the data set). Notice in the two examples the different contrast in the pictures. Figure 4.1 has a truck in the center on a highway, but the trailer of the truck gives off a bright spot. Figure 4.2 shows a road with no car, but the picture was taken in a shadow, thus part of the road is of a different shade then the rest. The idea is to get the training machine to recognize these differences and disregard them if need.

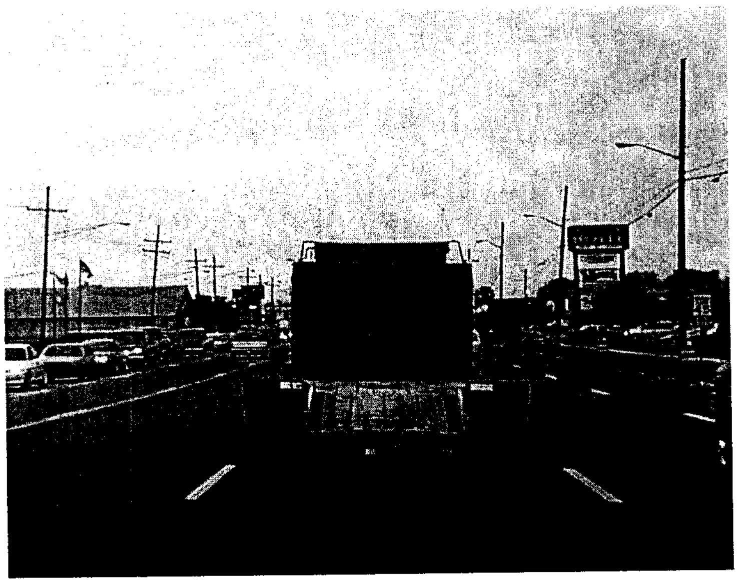

Figure 4.1

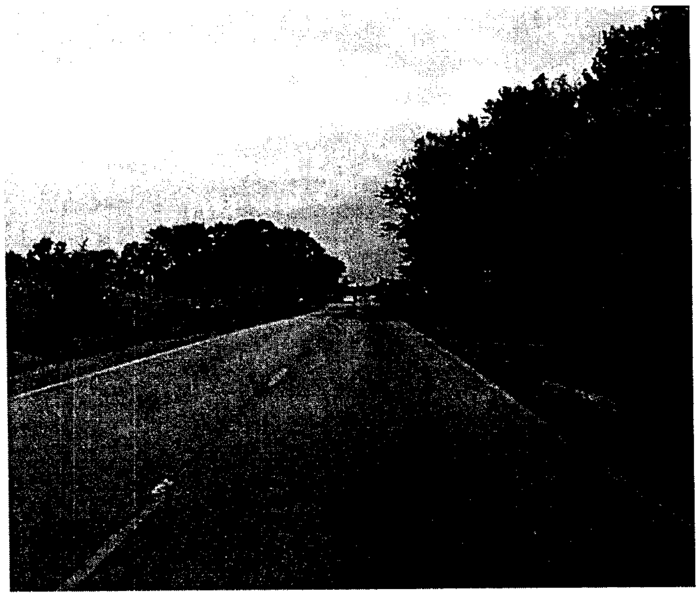

Figure 4.2

4.2 The Results

For the kernel I used a polynomial function of different degrees and compared them. I also compared different size training sets. Any data I didn't train with I used to test, so if the training set had 25 with a car and 25 with out a car, then the test set would be 50 with a car and 29 with out a car.

A classification matrix is shown using the rows as the class the picture belongs to. The columns represent what the image was classified as. Hence, row 1, column 1 represents a 'no car' classified correctly. row 2, column 1 represents a 'car' classified incorrectly.

Polynomial of degree 2.
Training set: Car = 25, No Car = 25.
Testing set: Car = 50, No Car = 29.

	Classified as Class -1	Classified as Class 1
Class -1 (No Car)	21	8
Class 1 (Car)	20	30

Classified Correctly: 64.5%

Polynomial of degree 3.
Training set: Car = 15, No Car = 15.
Testing set: Car = 60, No Car = 39.

	Classified as Class -1	Classified as Class 1
Class -1 (No Car)	21	8
Class 1 (Car)	9	41

Classified Correctly: 78.5%

Polynomial of degree 4.
Training set: Car = 15, No Car = 15.
Testing set: Car = 60, No Car = 39.

	Classified as Class -1	Classified as Class 1
Class -1 (No Car)	29	10
Class 1 (Car)	13	47

Classified Correctly: 76.8%

Training set: Car = 25, No Car = 25.
Testing set: Car = 50, No Car = 29.

	Classified as Class -1	Classified as Class 1
Class -1 (No Car)	20	9
Class 1 (Car)	6	44

Classified Correctly: 81.0%

Training set: Car = 35, No Car = 35.
Testing set: Car = 40, No Car = 19.

	Classified as Class -1	Classified as Class 1
Class -1 (No Car)	15	4
Class 1 (Car)	5	35

Classified Correctly: 84.7%

As you can see, the best results come from a polynomial kernel of degree 4 with a largest training set. It classified almost 85% correct, even with a small database of images from various types.

5.1 Final Thoughts on the Research Project

The data set used was a set of standard deviations from each row. I used this as a feature because I thought that the deviation of the gray scaled image rows would be a good measure of identifying if a car was present or not. However, when converting a color picture to a gray scale image, the darkness and bright ness scale is not linear. For example, if the gray scale range were from 255 to 1, the middle brightness would not be at 127. It would be around 200 (or 80, I forget which way the curve is bent). If it were linear, then I think that using the standard deviations as the feature vector would give better results.

Another approach might be to try to extract out the road from the rest of the data. If an object (matching the rest of the training set) were blocking the road then there would be a car. This is a much harder problem then I originally thought it would be. To be able to detect the road, I would need to be able to classify textures or perform the same learning as if there was a car. Classifying textures is beyond me at this time (and probably will always be), but with further investigation on feature spaces, it might be a solvable problem.

One final approach I thought might be relatively easy would be to use a box in a box approach. Start with the center of the picture and calculate the standard deviation. Then make a larger box and calculate the standard deviation of that. Continue until you end up with a box that encompasses the whole area of the image. Compare the standard deviations with the smallest box verse the next larger one. Then compare that one with its next larger box, etc. If there is a car then a certain size box and all the smaller ones should have relatively the same standard deviations verse the out side boxes. Unfortunately, these images have a lot of patches of darkness were their should be light, and a lot of patches of light were there should be darkness. These patches (or reversed spots) are caused by many things (like shadows in the road, light cars with dark roofs, reflection from wet pavement, a patch in the road, etc.). If a method could be found to outline the car and use that to compare a relative standard deviation (taking out reverse spots) then this might be a feasible alternative.

7.1 Optimization of the Support Vector Machine

At the start of this project, I had intended on coming up with a method for generalizing an optimization method. With an unusual amount of time spent on learning the theory behind support vector machines and a problem scheduling time with people to talk too about possible optimization categories to research, I have not been very successful. I have, however, come up with a few nuggets of information for the ambitious to pursue. Optimization of the SVMs can be accomplished by restricting the kernels to a low VC dimension. A low VC dimension will reduce the risk and, hopefully, increase the success of the kernel. Unfortunately, there are an uncountable infinite (maybe its countable) number of possible functions in each space. And a small VC dimension does not guarantee a good kernel (as shown in the car classification problem). I believe that optimization will have to be found in other methods. Feature extraction would be a good place to start.

8.1 Final Words on Support Vector Machines and this Project

Support vector machines are good classifiers for small amounts of data with out a distribution. many times, features can be less descriptive (thus, less computer intensive to generate the feature vector) then many other classification methods. This is the result of the learning process in the support vector machines.

This project showed that it is possible to get reasonable classification results from a diverse population. To increase the results to something that might be useable would take either a different data collection system (like using radar or sound rather than pictures) or a more complex feature extractor. Wavelets would be a good start as would texture detection.

9.1 References

- C. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1998.
- C. Burges. *A tutorial on Support Vector Machines for Pattern Recognition.* Data Mining and Knowledge Discovery, 1998.
- K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, 1990.
- S. Gunn. *Support Vector Machine for Classification and Regression.* ISIS Technical Report, 1998.
- R. Karlsen, D. Gorsich, G Gerhart. *Target Classification via Support Vector Machines.* Optical Engineering.
- B. Scholkopf, C Burges, A. Smola. *Advances in Kernel Methods. Support Vector Learning.* MIT Press, 1999.
- V. Vapnik. *Statistical Learning Theory.* Wiley, New York, 1998.

Appendix A

The following code was used to convert the images into the specified data set so that the processing of support vector machines could be done.

```
% convert_w_std.m program
% program to convert images std to a vector for SVM processing
% by Mike Del Rose

clear;

% assign values
NumOfImag_car = 75;  % number of images with a car
NumOfImag_nocar = 54;  % number of images without a car
shaveSide = 192; % the number of points to shave off on each side
shaveTopBot = 128; % the number of points to shave off the top and bottom.
newImageSize = [16 16]; % reduce by a factor of 16 (was 256x256)
imageFmt = 'jpg';

% *************** car images for training ***************
stdCarMatrix = [];
for i = 1:1:NumOfImag_car
   imageName = ['car' num2str(i) '.jpg'];

% read image - imageX = image, mapX = the color mapping
   imageX = imread(imageName,imageFmt);

% convert to gray scale
   grayImageX = rgb2gray(imageX);

% shave off the parts not needed
   sizeImageX = size(grayImageX);
   shavedImageX = grayImageX(shaveTopBot+1:sizeImageX(1)-shaveTopBot,
shaveSide+1:sizeImageX(2)-shaveSide);

% make the features the std of each row
   doubleImageX = double(shavedImageX);
   finalImageX = std(doubleImageX');

% store all vectors in 1 large matrix
   stdCarMatrix = [stdCarMatrix; finalImageX];
end;

% *************** no car images for training ***************
stdNoCarMatrix = [];
for i = 1:1:NumOfImag_nocar
   imageName = ['nocar' num2str(i) '.jpg'];

% read image - imageX = image, mapX = the color mapping
   imageX = imread(imageName,imageFmt);

% convert to gray scale
   grayImageX = rgb2gray(imageX);

% shave off the parts not needed
   sizeImageX = size(grayImageX);
   shavedImageX = grayImageX(shaveTopBot+1:sizeImageX(1)-shaveTopBot,
shaveSide+1:sizeImageX(2)-shaveSide);

% resize to make a more useable matrix using nearest neighbor method
   doubleImageX = double(shavedImageX);
   finalImageX = std(doubleImageX');

% store all vectors in 1 large matrix
   stdNoCarMatrix = [stdNoCarMatrix; finalImageX];
end;

% save final matrices
   save 'std_car_matrix.mat' stdCarMatrix;
   save 'std_nocar_matrix.mat' stdNoCarMatrix;
```

Appendix B

The following code was used to take in a data set and use the support vector machines to train and classify the data.

```
% This program is used to test out ideas in SVM.
% SVM functions used will be svctrain, and svcplotc.
% Mike Del Rose
%

clear all;

% Initialize defined variables:
ON = 1;
OFF = 0;
%carMatrixName = ['Final_car_matrix.mat'];
%noCarMatrixName = ['Final_nocar_matrix.mat'];
carMatrixName = ['std_car_matrix.mat'];
noCarMatrixName = ['std_nocar_matrix.mat'];
NumOfTrainCar = 25;
NumOfTrainNoCar = 25;

% set debug flags
DEBUG_DATA = OFF;
DEBUG_VAR = OFF;
DEBUG_PRINT_DATA = OFF;
DEBUG = OFF;

% Set Normalization to on or off (mean = 0, sigma = 1)
NORMALIZE = ON;

% set up data set by reading in data.
   cd ../project
%   eval(['load ' carMatrixName ' finalCarMatrix']);
%   eval(['load ' noCarMatrixName ' finalNoCarMatrix']);
   eval(['load ' carMatrixName ' stdCarMatrix']);
   eval(['load ' noCarMatrixName ' stdNoCarMatrix']);

%   X1_Final = double(finalCarMatrix);  % make sure it is double format (for STD)
%   X2_Final = double(finalNoCarMatrix);  % make sure it is double format (for STD)
   X1_Final = double(stdCarMatrix);  % make sure it is double format (for STD)
   X2_Final = double(stdNoCarMatrix);  % make sure it is double format (for STD)
   Y1_Final = ones(size(X1_Final,1),1);
   Y2_Final = -ones(size(X2_Final,1),1);

   cd ../svm_code
 % Normalize data (if set to ON)
   if (NORMALIZE)
      stdX1 = std(X1_Final');
      stdX2 = std(X2_Final');
      meanX1 = mean(X1_Final');
      meanX2 = mean(X2_Final');
      sizeX1 = size(X1_Final);
      sizeX2 = size(X2_Final);
      for i = 1:1:sizeX1(2)
         X1_Norm(:,i) = X1_Final(:,i) - meanX1';
         X1_Norm1(:,i) = X1_Norm(:,i) ./ stdX1';
       end;
      for i = 1:1:sizeX2(2)
          X2_Norm(:,i) = X2_Final(:,i) - meanX2';
          X2_Norm1(:,i) = X2_Norm(:,i) ./ stdX2';
       end;
      % reset the current values of the data
      clear X1_Final X2_Final;
      X1_Final = X1_Norm1;
      X2_Final = X2_Norm1;
   end;

   if (DEBUG_DATA)  X1_Final, X2_Final, Y1_Final, Y2_Final, end

% seperate training and testing data
sizeX1 = size(X1_Final);
sizeX2 = size(X2_Final);
X1_Train = X1_Final(1:NumOfTrainCar,:);
Y1_Train = Y1_Final(1:NumOfTrainCar);
X2_Train = X2_Final(1:NumOfTrainNoCar,:);
Y2_Train = Y2_Final(1:NumOfTrainNoCar);

X1_Test = X1_Final(NumOfTrainCar+1:sizeX1(1),:);
Y1_Test = Y1_Final(NumOfTrainCar+1:sizeX1(1));
X2_Test = X2_Final(NumOfTrainNoCar+1:sizeX2(1),:);
Y2_Test = Y2_Final(NumOfTrainNoCar+1:sizeX2(1));


% Combine classes together to make training and testing set
X = [X1_Train; X2_Train];
Y = [Y1_Train; Y2_Train];

X_Test = [X1_Test; X2_Test];
Y_Test = [Y1_Test; Y2_Test];

if (DEBUG_DATA)  X, Y, X_Test, Y_Test, end;

if (DEBUG_PRINT_DATA)
   xmin = -5; xmax = 12; ymin = -2; ymax = 16;
   axis normal
   axis([xmin, xmax, ymin, ymax])
   title('Data plot')
   hold on
   plot(X1_Final(:,1),X1_Final(:,2),'bx',X2_Final(:,1),X2_Final(:,2),'r+')
   X_axis = [xmin:.1:xmax]';
   Y_axis = [ymin:.1:ymax]';
   plot(X_axis,0,'k--',0,Y_axis,'k--')
else % continue normal execution
   % initialize variables
   ker = 'poly';
   C = inf;  % upper bound for non-seperable case
   x00 = 0;  % starting value
   p1 = 3; % parameter 1 of the kernel function
   p2 = 0; % parameter 2 of the kernel function
   aspect = 0;  % Aspect ratio (0 = fixed; 1 = variable)
   mag = 1;  % magnification of the display
   xaxis = 1;  % xaxis input column X (default = 1)
   yaxis = 2;  % yaxis input column X (default = 2)

   if (DEBUG)
      X, xaxis
      fprintf ('min(X(:,xaxis))' ), min(X(:,xaxis))
      fprintf ('max(X(:,xaxis))' ), max(X(:,xaxis))
      yaxis
      fprintf ('min(X(:,yaxis))' ), min(X(:,yaxis))
      fprintf ('max(X(:,yaxis))' ), max(X(:,yaxis))
   end

   param = [C,x00,p1,p2];
   params = [aspect,mag,xaxis,yaxis,p1,p2];

   if (DEBUG_VAR)  ker, param, params, end

   % train data set
   [nsv, alpha, bias, svi] = svctrain(X,Y,ker,param);
   if (DEBUG)  nsv, alpha, bias, param, end

   %Plot the results
   svcplotc(X,Y,ker,alpha,bias,params)
 end % if not in DEBUG_PRINT_DATA mode
```

```
% test of the data set.
trainSet = X;
trainY = Y;
testSet = X_Test;
p = [p1 p2];
finalTest = svctest(ker, alpha, bias, trainSet, testSet, trainY, p);

% Compute classification
classMatrix = [0 0;0 0];
testVector = sign(finalTest);
sizeTV = size(testVector);
for i = 1:1:sizeTV(1)
   if testVector(i) == Y_Test(i)
      if Y_Test(i) == 1
         classMatrix(2,2) = classMatrix(2,2) + 1;  % classified class 1 correctly
      else
         classMatrix(1,1) = classMatrix(1,1) + 1;  % classified class -1 correctly
      end;
   else
      if Y_Test(i) == 1
         classMatrix(2,1) = classMatrix(2,1) + 1; % classified class 1 incorrectly
      else
         classMatrix(1,2) = classMatrix(1,2) + 1; % classified class -1 incorrectly
      end
   end;
end;

classMatrix
output1 = ['for class 1, ' num2str(classMatrix(2,2)) ' were classified correctly.  '
num2str(classMatrix(2,1)) ' were classified incorrectly.'];
output2 = ['for class -1, ' num2str(classMatrix(1,1)) ' were classified correctly.  '
num2str(classMatrix(1,2)) ' were classified incorrectly.'];
output1, output2
```